LiveKit Agents で作るサーバサイド WebRTC AI エージェント

2024/10/22
WebRTC Meetup Tokyo 28
こーのいけ

Whoami

こーのいけ

X: ko_noike　GitHub他: kounoike

色々やってるフリーランスエンジニア

IoT/画像処理/音声処理/映像処理/リアルタイムコミュニケーション

最近やったこと

国内のPerlコードゴルフコンペに参加。半日強でトップをかっさらう

→ https://zenn.dev/kounoike/articles/20241007-yapcjapan2024-perlbatross

```
#!perl -lap
$_=%{{map{join(utf8'decode($_),sort/./g),1}@F}}>1^1
```

```
#!perl -p
$n=<>;s!..!$_=$&;$n=~/../g;$_.=$&;/.(.)\1\1/||/(.)/;$1!ge
```

サーバサイド AI エージェント

Wasm/WebGPUなどでクライアントサイド（ブラウザ）でのAI活用が盛ん

→クライアントサイドでの処理が向かないユースケース

- クライアントの処理能力が貧弱
- 推論モデルをクライアントに配布したくない
- 偽装クライアントで処理を誤魔化されると困る（監視など）

→サーバサイドでの推論処理が向いている

（「サーバ」と言っているが、WebRTC SFUから見ればクライアント）

サーバサイドAIエージェントの実装

- ネイティブのWebRTCクライアントライブラリ
  - 受信ストリームの映像からフレーム、音声からサンプルの取得
  - 推論結果に基づいた変換後のストリームの送信
  - DataChannelによるデータ送信
- ルーム作成・ユーザの接続・切断などをトリガーにしたWebhook等
  - ジョブの起動管理
- 水平スケーリング

LiveKit Agents

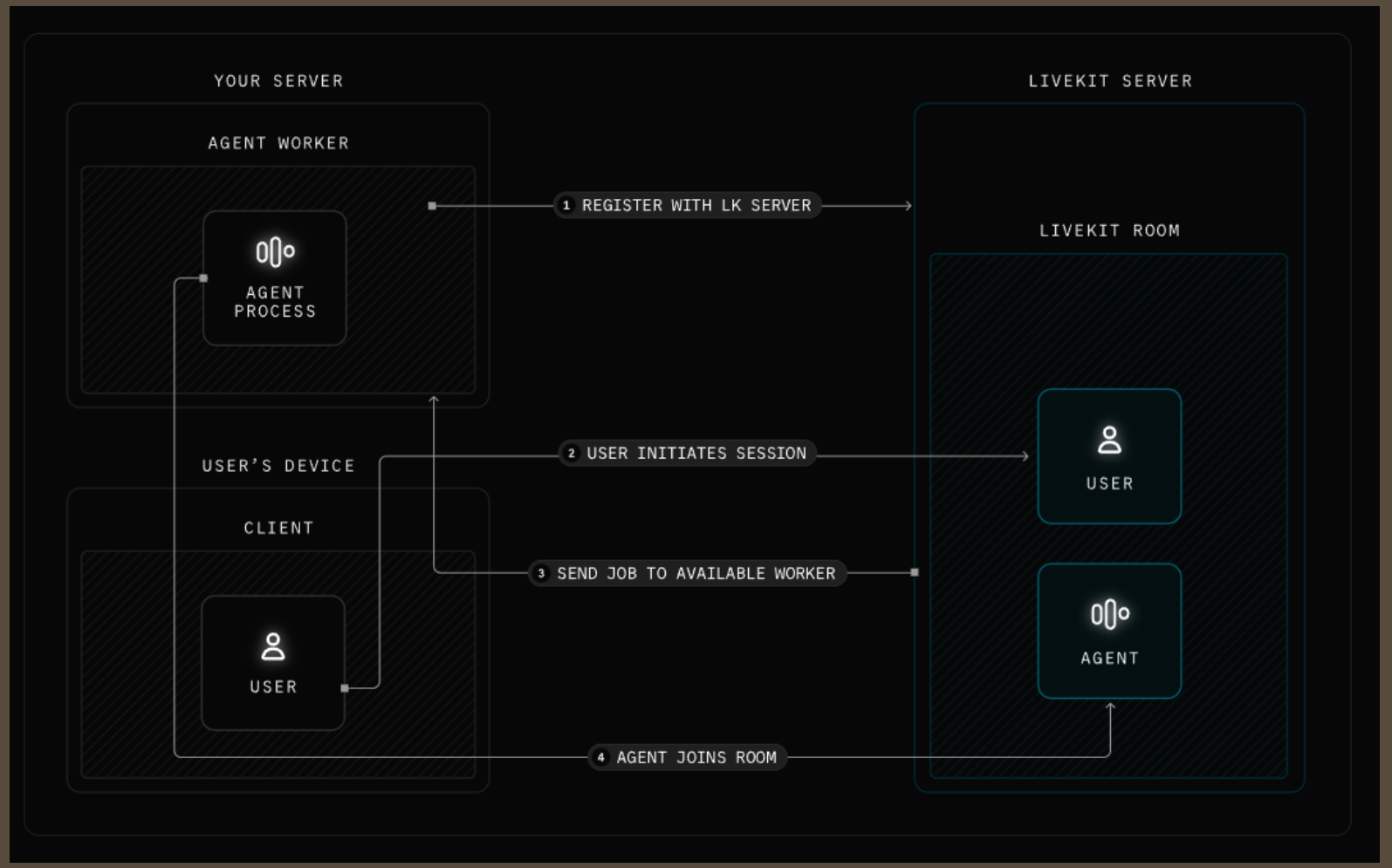
- LiveKit （SFU/クラウドSFUサービス）が提供しているAIエージェント作成用ライブラリ
  - 今のところ Python, JavaScript
- SFUサーバ側にエージェント管理の機能、それに対するクライアント実装
  →Webhookなどが不要。エージェント側でグローバルにポート開ける必要なし
- ルームの作成 or ユーザの参加をトリガーに対応するジョブを起動
- エージェントのスケーリングに対応
  - ワーカーを複数起動すればサーバ側が自動的にジョブを割り振ってくれる
- AIエージェントで良く使う機能を「プラグイン」として提供
- 流行りのOpenAI Realtime APIへの接続例もあり

LiveKit 情報アップデート

2024/08 料金プランの変更

- 有料プランに月額最低価格が追加
  - $50/mo, $500/mo
- 基本的に転送量のみの従量課金→転送量＆接続時間
  - 転送量単価は値下げ
- 同時接続数制限
  - $50/moプラン: 1000、$500/moプラン: 無制限

エージェントの構成

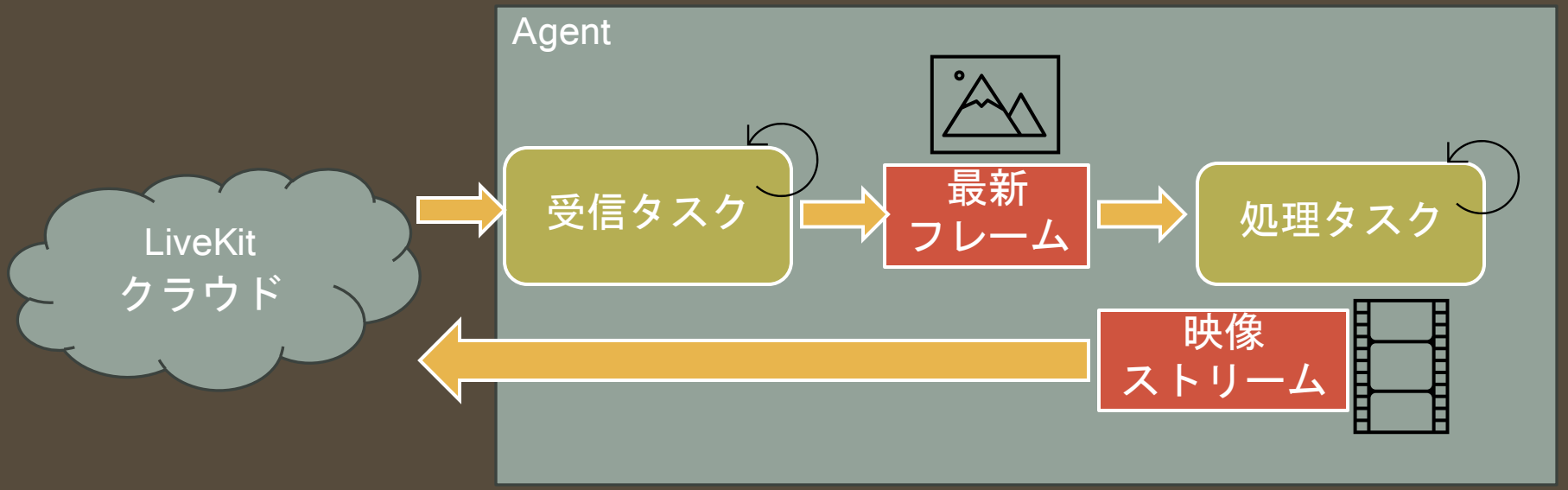


1. エージェントの起動
2. ユーザの参加
3. ジョブの送信
4. エージェントの参加

公式ドキュメントより

コード例: 顔検出

- 映像から顔検出して四角を描画して送り返す
- 音声は処理しない
- 1ルーム1ユーザが前提
- OpenCVの顔検出（FacerDetectorYN: CPU推論）
- 全フレーム処理する性能が出ないので、間に合わないフレームは無視



```python
async def entrypoint(ctx: JobContext):
    '''ジョブのエントリーポイント'''
    # ジョブ初期化処理
    source = rtc.VideoSource(WIDTH, HEIGHT)
    publish_track = rtc.LocalVideoTrack.create_video_track("send stream", source)
    frames = {"cur": np.zeros((HEIGHT, WIDTH, 3), dtype=np.uint8)}
    # 接続
    await ctx.connect(auto_subscribe=AutoSubscribe.VIDEO_ONLY)
    # 参加者を待つ
    await ctx.wait_for_participant()

    async def process_video_track(participant: rtc.RemoteParticipant, track: rtc.Track):
        '''受信映像トラック処理'''

    # トラック購読時のハンドラ
    @ctx.room.on("track_subscribed")
    def on_track_subscribed(track: rtc.Track, publication: rtc.TrackPublication, participant: rtc.RemoteParticipant):
        if track.kind == rtc.TrackKind.KIND_VIDEO:
            asyncio.create_task(process_video_track(participant, track))

    async def detect_faces():
        '''顔検出処理'''

    face_detect_task = asyncio.create_task(detect_faces())
    await ctx.room.local_participant.publish_track(publish_track, rtc.TrackPublishOptions(source=rtc.TrackSource.SOURCE_CAMERA))

    # 参加者切断時のハンドラ
    @ctx.room.on("participant_disconnected")
    def on_participant_disconnected(participant: rtc.RemoteParticipant):
        face_detect_task.cancel()

if __name__ == "__main__":
    '''起動処理（エージェントの登録） 接続先やクレデンシャルは環境変数から取得'''
    cli.run_app(WorkerOptions(entrypoint_fnc=entrypoint))
```

```python
async def process_video_track(participant: rtc.RemoteParticipant, track: rtc.Track):
    # rtc.VideoStreamにしてフレーム受信イベントごとに処理
    video_stream = rtc.VideoStream(track)
    async for event in video_stream:
        # YUV I420で来るのでBGR(OpenCVなので…)に変換
        yuv_frame = np.frombuffer(event.frame.data, dtype=np.uint8).reshape((event.frame.height * 3 // 2, event.frame.width))
        current_frame = cv2.cvtColor(yuv_frame, cv2.COLOR_YUV2BGR_I420)
        # 処理タスクと共有する最新画像を更新
        frames["cur"] = current_frame
        logging.info(f"got frame {current_frame.shape}")

@ctx.room.on("track_subscribed")
def on_track_subscribed(track: rtc.Track, publication: rtc.TrackPublication, participant: rtc.RemoteParticipant):
    if track.kind == rtc.TrackKind.KIND_VIDEO:
        asyncio.create_task(process_video_track(participant, track))
```

```python
async def detect_faces():
    # FaceDetectorYNを作る
    detector = cv2.FaceDetectorYN.create("face_detection_yunet_2023mar.onnx", "", (320, 180))

    # 処理はずっと繰り返す
    while True:
        # 受信タスクと共有する最新画像を取得
        current_frame = frames["cur"]
        # 顔検出器に画像サイズを設定
        detector.setInputSize((current_frame.shape[1], current_frame.shape[0]))
        start = time.time()
        # 顔検出の実行
        _, faces = detector.detect(current_frame)
        end = time.time()
        logging.info(f"faces: {faces} duration: {end - start}")
        # 結果描画用フレームを作成（LiveKitのストリームに与えるためRGBA形式）
        next_frame = cv2.cvtColor(current_frame, cv2.COLOR_BGR2RGBA)
        # 顔領域の描画
        if faces is not None:
            for face in faces:
                x0, y0, w, h = face[:4]
                x1, y1 = x0 + w, y0 + h
                logging.info(f"face: {x0}, {y0}, {x1}, {y1}")
                cv2.rectangle(next_frame, (int(x0), int(y0)), (int(x1), int(y1)), (255, 255, 255, 255), 2)
        # 送信ストリームのフレームを作成
        frame = rtc.VideoFrame(next_frame.shape[1], next_frame.shape[0], rtc.VideoBufferType.RGBA, next_frame.tobytes())
        # 送信ストリームにフレームをキャプチャ
        source.capture_frame(frame)
        # asyncioが他のタスクに処理を譲るために少しスリープ
        await asyncio.sleep(0.001)
```

コード例: 音声エージェント

- LiveKitのexamples
- 音声パイプラインエージェント

VAD(Voice Activity Detection)

→STT(Speech-To-Text)

→LLM(Large Language Model)

→TTS(Text-To-Speech)

```python
async def entrypoint(ctx: JobContext):
    """This example demonstrates a VoicePipelineAgent that uses OpenAI's Assistant API as the LLM"""
    initial_ctx = llm.ChatContext()

    await ctx.connect(auto_subscribe=AutoSubscribe.AUDIO_ONLY)
    participant = await ctx.wait_for_participant()

    # When you add a ChatMessage that contain images, AssistantLLM will upload them
    # to OpenAI's Assistant API.
    # It's up to you to remove them if desired or otherwise manage them going forward.
    def on_file_uploaded(info: OnFileUploadedInfo):
        logger.info(f"{info.type} uploaded: {info.openai_file_object}")

    agent = VoicePipelineAgent(
        vad=silero.VAD.load(),
        stt=deepgram.STT(),
        llm=AssistantLLM(
            assistant_opts=AssistantOptions(
                create_options=AssistantCreateOptions(
                    model="gpt-4o",
                    instructions="You are a voice assistant created by LiveKit. Your interface with users will be voice.",
                    name="KITT",
                )
            ),
            on_file_uploaded=on_file_uploaded,
        ),
        tts=openai.TTS(),
        chat_ctx=initial_ctx,
    )
    agent.start(ctx.room, participant)
    await agent.say("Hey, how can I help you today?", allow_interruptions=False)


if __name__ == "__main__":
    cli.run_app(WorkerOptions(entrypoint_fnc=entrypoint))
```

初期化
接続して
参加者待ち

パイプライン
作成

起動して
最初の一言

気になるところ

- スケーリングさせたときのVM増加等のトリガ
  - インフラ側との関係が強いので、そっち側で何とかする？
- VMを減らすとき/デプロイ時の断絶
  - 「接続済みは処理を続けるが、新規接続を受け付けない」みたいなモードあるか？
  - 接続済みの処理が全部終わったら終了、みたいなことが出来るか？
- 推論リソース（GPU）の効率的活用
  - 推論部分を別サーバにした方がいいかも
- エンコード・デコードのHWA
  - LiveKitのクライアントライブラリ自体にHWA対応が入ってないような...？